Technical Communications of the International Conference on Logic Programming, 2010 (Edinburgh), pp. 54–63
http://www.floc-conference.org/ICLP-home.html

INDUCTIVE LOGIC PROGRAMMING AS ABDUCTIVE SEARCH

DOMENICO CORAPI [1] AND ALESSANDRA RUSSO [1] AND EMIL LUPU [1]

[1] Department of Computing
Imperial College London
180 Queen's Gate, SW7 2AZ
London, UK
E-mail address: {d.corapi,a.russo,e.c.lupu}@ic.ac.uk

Abstract. We present a novel approach to non-monotonic ILP and its implementation called TAL (Top-directed Abductive Learning). TAL overcomes some of the completeness problems of ILP systems based on Inverse Entailment and is the first top-down ILP system that allows background theories and hypotheses to be normal logic programs. The approach relies on mapping an ILP problem into an equivalent ALP one. This enables the use of established ALP proof procedures and the specification of richer language bias with integrity constraints. The mapping provides a principled search space for an ILP problem, over which an abductive search is used to compute inductive solutions.

Introduction

Inductive Logic Programming (ILP) [Lav94] is a machine learning technique concerned with the induction of logic theories from positive and negative examples and has been successfully applied to a wide range of problems [Dŏŏ]. Its main virtue, the highly expressive representation language, is also the cause of its high computational complexity. Some ILP systems attempt to efficiently find a less then perfect hypothesis by using heuristics to navigate the search space effectively [Qui96], [Ric95]. Others focus on completeness and aim for perfect accuracy with respect to the examples, searching the space thoroughly for an optimal solution. Among these XHAIL [Ray09a] has identified Abductive Logic Programming (ALP) [Kak92] as a means to deal with incomplete theories and provide semantics for negation as failure (NAF) [Cla77]. XHAIL, like other *inverse entailment* (IE) based systems, abductively derives a lower bound for the search space that is then generalised. In contrast, *Top-down* ILP systems like [Mug08, Bra99, Bos94] construct the hypothesis by specialising an overly general theory without a lower bound. However existing top-down systems limit the expressiveness of the language and the possible outcome of the learning (e.g. concepts learned must be observed in the training data, recursion is not allowed and the use of negation is limited).

Abductive proof procedures have been extensively employed as part of ILP systems (e.g. [Esp00]) or extended for inductive reasoning (e.g. [Adé95]). In contrast to these existing approaches, we propose a novel mechanism that *maps an ILP problem into an equivalent ALP*

Key words and phrases: Inductive Logic Programming, Abductive Logic Programming, Non-monotonic Reasoning.

Technical Communications of the 26th International Conference on Logic Programming, Edinburgh, July, 2010
Editors: Manuel Hermenegildo, Torsten Schaub
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.ICLP.2010.54

instance. An ILP task is thus translated into an ALP problem whose solution is translated back into a solution of the original problem. The resulting top-down ILP system, called TAL (Top-directed Abductive Learning), offers several advantages over existing techniques. TAL is able to handle negation within the learning process and is able to learn non-monotonic hypotheses, relying on the semantics of the underlying abductive proof procedure employed; allows expressive language bias specifications that subsume mode declarations and can be combined with integrity constraints; performs non-observational [Moy03] and multiple predicate learning [Mal98]; and makes use of constraint solving techniques. Non-monotonic ILP has been successfully applied to bioinformatics [Ray08] and requirement engineering [Alr09] and as showed in [Cor09] and [Ray09b] can also be employed to perform theory revision.

In the particular case of definite theories, the TAL search space includes hypotheses that are not found by established Inverse Entailment based systems like PROGOL [Mug95] or ALECTO [Moy03] and provides a more effective solution to learning interdependent concepts compared to the state of art ILP systems, e.g. [Kim09, Ray09a]. Though not explored in depth here, the principled search space characterised by ALP includes abductive solutions that represent partial inductive solutions, which can be measured in terms of some scoring function (e.g. accuracy on the given set of examples) thereby enabling the use of heuristic based search strategies.

The paper is organised as follows. First, we introduce the notation and relevant background concepts. We then describe the representation underlying the learning system and discuss the learning mechanism. Then, we present through examples some of the main features of the system, and discuss related work. We conclude with final remarks and directions for future work.

1. Abductive and Inductive Logic Programming

ALP and ILP are extensions of logic programming. They both search for a hypothesis that is able to account for some given evidence. ALP constructs hypotheses in the form of ground facts. ILP systems generate rules that are able to discriminate between positive and negative examples that represent the training data. In general, ILP is regarded as a machine learning technique and used when a certain knowledge base must be enriched with rules that are also able to classify new examples. We assume in the following that the reader is familiar with first-order logic and logic programming [Llo87]. Following Prolog [Sha94] conventions, predicates, terms and functions are represented with an initial lower case letter and variables are represented with an initial capital letter.

Definition 1.1. An *ALP task* is defined as $\langle g, T, A, I\rangle$ where T is a normal logic program, A is a set of *abducible* facts, I is a set of *integrity constraints* and g is a ground goal. $\Delta \in ALP\langle g, T, A, I\rangle$ is an abductive solution for the ALP task $\langle g, T, A, I\rangle$, if $\Delta \subseteq A$, $T \cup \Delta$ is consistent, $T \cup \Delta \models g$ and $T \cup \Delta \models I$. $ALP\langle g, T, A, I\rangle$ denotes the set of all abductive solutions for the given ALP task.

Note that the abductive task, as defined, deals with ground goals, thus being a specific case of the setting in [Kak92]. The notion of entailment is not fixed since, as discussed later, the approach proposed in this paper is not committed to a particular semantics.

Definition 1.2. An *ILP task* is defined as $\langle E, B, S\rangle$ where E is a set of ground positive or negative literals, called *examples*, B is a *background theory* and S is a set of clauses called *language bias*. The theory $H \in ILP\langle E, B, S\rangle$, called *hypothesis*, is an inductive solution

for the task $\langle E, B, S\rangle$, if $H \subseteq S$, H is consistent with B and $B \cup H \models E$. $ILP\langle E, B, S\rangle$ denotes the set of all inductive solutions for the given task.

We consider the case where B and H are normal logic programs and E is a set of ground literals (with positive and negative ground literals representing positive and negative examples, respectively).

The space of possible solutions is inherently large for all meaningful applications so different levels of constraints are imposed to restrict the search for hypotheses.When possible, besides the background knowledge about the modelled world, some *a priori* knowledge on the structure of the hypothesis can be employed to impose an instance-specific *language bias* S. Mode declarations are a common tool to specify a language bias.

Definition 1.3. A *mode declaration* is either a head or body declaration, respectively $modeh(s)$ and $modeb(s)$ where s is called a *schema*. A schema s is a ground literal containing placemarkers. A *placemarker* is either '$+type$' (input), '$-type$' (output), '$\#type$' (ground) where *type* is a constant.

Given a schema s, s^* is the literal obtained from s by replacing all placemarkers with different variables $X_1, ..., X_n$; $\texttt{type}(\texttt{s}^*)$ denotes the conjunction of literals $t_1(X_1), ..., t_n(X_n)$ such that t_i is the type of the placemarker replaced by the variable X_i; $\texttt{ground}(\texttt{s}^*)$ is the list of the variables that replace the ground placemarkers in s, listed in order of appearance left to right. Similarly $\texttt{inputs}(\texttt{s}^*)$ and $\texttt{outputs}(\texttt{s}^*)$ are, respectively, the lists of the variables replacing input and output placemarkers in s. For example, for mode declaration $m3$ in Sec. 3, $s = even(+nat)$, $s^* = even(X)$, $\texttt{type}(\texttt{s}^*) = nat(X)$, $\texttt{outputs}(\texttt{s}^*) = \texttt{ground}(\texttt{s}^*) = []$, $\texttt{inputs}(\texttt{s}^*) = [X]$.

A rule r is *compatible* with a set M of mode declarations iff (a) there is a mapping from each head/body literal l in r to a corresponding head/body declaration $m \in M$ with schema s and l is subsumed by s^*; (b) each output placemarker is bound to an *output variable*; (c) each input placemarker is bound to an output variable appearing in the body before or to a variable in the head; (d) each ground placemarker is bound to a ground term; (e) all variables and terms are of the corresponding type. The use of head variables in output mode declarations is not discussed here for space constraints.

In the next sections the language bias S is specified in terms of a set M of mode declarations, and denoted as $s(M)$. Each mode declaration $m \in M$ is uniquely identified by a label id_m, called *mode declaration identifier*.

2. TAL

An ILP problem can be seen as a search for a hypothesis where we choose how many rules to use and for each rule which predicates to use in the head and in each of the body conditions and how many body conditions are used. Additionally, different choices are possible on how arguments are unified or grounded. In this section, we present the mapping of a set M of mode declarations into a *top theory* $\top$ that constrains the search by imposing a generality upper bound on the inductive solution. An abductive proof procedure is instantiated on this top theory together with the background theory. The abductive derivation identifies the heads of the rules (of a hypothesis solution) and the conditions needed to cover positive examples and exclude negative examples, ensuring consistency. The abductive solution is guaranteed to have a corresponding inductive hypothesis H that is a solution with respect to the examples.

2.1. Hypothesis representation

We use a list-based representation to encode an inductive solution, as a set of rules, into a set of facts by mapping each literal in the clauses into instances of its corresponding mode declaration. This allows the representation of rules as abducible facts. An inductive hypothesis $H \subseteq s(M)$ is composed of a set of rules $\{r_1, ..., r_n\}$. Each rule r is associated with an *output list*, i.e. an ordered list of the variables in r that replace output placemarkers in body literals or input placemarkers in the head. The *output identifier* associated with each of these variables is the position of the variable in the output list. Given a set M of mode declarations, each rule r of the form $l_1 \leftarrow l_2, ..., l_n$, compatible with M, can be represented as an ordered list $l = [\mathtt{l_1}, \mathtt{l_2}, ..., \mathtt{l_n}]$ where each $\mathtt{l_i}$ is a tuple $(id_m, [c_1, ..., c_p], [o_1, ..., o_q])$; id_m is the identifier of the mode declaration in M that l_i maps to; each c_j is a ground term replacing a ground placemarker in the mode declaration identified by id_m; each o_k is an output identifier and encodes the fact that the k^{th} input variable in l_i (in order of appearance left to right) unifies with the variable indicated by o_k. We refer to this transformation of a rule r into a list l as the function $l = t_M(r)$.

Example 2.1. Given the following three mode declarations $M = \{m1 : modeh(p(+any)),$ $m2 : modeb(q(+any, \#any)),$ $m3 : modeb(q(+any, -any))\}$ the rule $r = p(X) \leftarrow q(X, Y), q(Y, a)$, compatible with M, is associated to the output list $[X, Y]$ where X replaces the input placemarker in $m1$ and Y replaces the output placemarker in $m3$. The output identifier of X is the integer 1 and the output identifier of Y is 2. r is represented as the list $l = [(m1, [], []), (m3, [], [1]), (m2, [a], [2])] = t_M(r)$. The first element of the list associates the head $p(X)$ to mode declaration $m1$. The second element associates the condition $q(X, Y)$ to mode declaration $m3$ and links the first and only input variable to X. The third element of the list associates the condition $q(Y, a)$ to mode declaration $m2$, links the input variable to Y and instantiates the second argument to a.

It is easy to see that every possible rule within $s(M)$ can be encoded according to this representation. Also, it is always possible to derive a unique rule r from a well-formed list l. We refer to this transformation as $r = {t_M}^{-1}(l)$.

Rules in a final inductive hypothesis theory H, are associated with a unique *rule identifier* r_{id}, an integer from 1 to the maximum number, MNR, of rules allowed in H. The abductive representation $\Delta = T_M(H)$ of an hypothesis theory H, is the set of facts $rule(r_{id}, l) \in \Delta$, one for each rule $r \in H$, such that (a) r_{id} is the rule identifier of r; and (b) $l = t_M(r)$. The inverse transformation $H = {T_M}^{-1}(\Delta)$ is similarly defined.

2.2. Mapping mode declarations into an abductive top theory

The first computational step in TAL is the generation of a top theory $\top$ from a set M of mode declarations, as defined below, where *prule* and *rule* are abducible predicates.

Definition 2.2. Given a set M of mode declarations, $\top = f(M)$ is constructed as follows:

- For each head declaration $modeh(s)$, with unique identifier id_m, the following clause is in $\top$

$$s^* \leftarrow \mathtt{type}(\mathtt{s}^*), prule(RId, [id_m, \mathtt{ground}(\mathtt{s}^*), []]), rule\_id(RId), \\ body(RId, \mathtt{inputs}(\mathtt{s}^*), [id_m, \mathtt{ground}(\mathtt{s}^*), []]) \tag{2.1}$$

- The following clause is in $\top$

$$body(RId, \_, Rule) \leftarrow rule(RId, Rule) \tag{2.2}$$

- For each body declaration $modeb(s)$, with identifier id_m the following clause is in $\top$

$$body(RId, Inputs, Rule) \leftarrow append(Rule, [(id_m, \texttt{ground}(\texttt{s}^*), Links)], NRule), \\ prule(RId, NRule), link\_variables(inputs(s^*), Inputs, Links), s^*, \\ \texttt{type}(\texttt{s}^*), append(Inputs, \texttt{outputs}(\texttt{s}^*), Outputs), body(RId, Outputs, NRule) \tag{2.3}$$

As previously defined, s^* contains newly defined variables instead of placemarkers in the schema s. Since the abductive derivation is instantiated on $B \cup \top$, for all predicates appearing in head mode declarations the procedure can both use the partial definition in B or learn a new clause, unifying the current goal with s^*. s^* can also be a negative condition corresponding to a body mode declaration whose schema is a negative literal. $rule\_id(RId)$ is true whenever $1 \leq RId \leq MNR$.

$body(r, i, c)$ keeps track of the rules that are built and provides the choice of extending a partial rule (rule (2.3)) or delivering a rule as final (rule (2.2)). The first argument is the rule identifier; the second is the list of the outputs collected from the literals already added to the rule; the third is the list representing a partial rule. $link\_variables([a_1, ..., a_m], [b_1, ..., b_n], [o_1, ..., o_m])$ succeeds if for each element in the first list a_i, there exist an element in the second list b_j such that a_i unifies with b_j and $o_i = j$. $append(l_1, l_2, l_3)$ has the standard Prolog definition [Sha94]. The abducible *prule* is used to control the search through integrity constraints. For example, if we are not interested in searching for rules in which two mode declarations i and j appear together in the body, then an integrity constraint of the type $\leftarrow prule(\_, R1), member((i, \_, \_), R1), member((j, \_, \_), R1)$ can be added to I to prune such solutions in the abductive search.

In order to maintain the well-formedness of our rule's encoding and avoid trivial states of the search, a set of fixed integrity constraints I_f (omitted for brevity) is used in the abductive search.

2.3. Learning

Definition 2.3. Given an ILP task $\langle E, B, M \rangle$, $H = {T_M}^{-1}(\Delta)$ is an inductive solution derived by TAL iff $\Delta \in ALP\langle g, B \cup \top, I, A \rangle$ where $\top = f(M)$, g is the negated conjunction of the literals in E, I is a set of integrity constraints that includes I_f and $A = \{rule/2, prule/2\}$

Procedurally, an initial translation produces the ALP task introducing the new theory $\top$. An abductive proof procedure derives the abductive hypothesis Δ that is then transformed into the final inductive solution.

Theorem 2.4. *Let us consider a theory B, a set M of mode declarations, a conjunction of (possibly negated) literals E, and a set H of rules, such that $H \subseteq s(M)$. Then $B \cup H \vdash_{SLDNF} E$ iff $B \cup \top \cup \Delta \vdash_{SLDNF} E$, where $\top = f(M)$ and $\Delta = T_M(H)$*

As corollaries of Theorem (2.4), it is possible to establish soundness and completeness of our TAL system based on the properties of the underlying abductive system and on the soundness and completeness of SLDNF w.r.t. the semantics.

3. Example

The following is a modified version of the well-known example proposed in [Yam97], where even and odd numbers are both target concepts and learnt from three examples of odd numbers. The *even* predicate is partially defined, i.e. base case $even(0)$.

$even(X) \leftarrow prule(RId, [(m1, [], []]),$
$\quad nat(X), rule\_id(RId),$
$\quad body(RId, [X], [(m1, [], []])$

$odd(X) \leftarrow prule(RId, [(m2, [], []]),$
$\quad nat(X), rule\_id(RId),$
$\quad body(RId, [X], [(m2, [], []])$

$body(RId, \_, Rule) \leftarrow rule(RId, Rule)$

$body(RId, Inputs, Rule) \leftarrow$
$\quad append(Rule, [(m3, [], Links)], NRule),$
$\quad prule(RId, NRule),$
$\quad link\_variables(X, Inputs, Links),$
$\quad even(X), nat(X), body(RId, Inputs, NRule)$

$body(RId, Inputs, Rule) \leftarrow$
$\quad append(Rule, [(m4, [], Links)], NRule),$
$\quad prule(RId, NRule),$
$\quad link\_variables(X, Inputs, Links),$
$\quad odd(X), nat(X), body(RId, Inputs, NRule)$

$body(RId, Inputs, Rule) \leftarrow$
$\quad append(Rule, [(m5, [], Links)], NRule),$
$\quad prule(RId, NRule),$
$\quad link\_variables(X, Inputs, Links), X = s(Y),$
$\quad nat(X), nat(Y), append(Inputs, [Y], Outputs),$
$\quad body(RId, Outputs, NRule)$

Figure 1: Top theory $\top$ for the even-odd example.

$$B = \left\{ \begin{array}{l} even(0) \\ nat(0) \\ nat(s(X)) \leftarrow nat(X) \end{array} \right. \qquad M = \left\{ \begin{array}{l} m1 : modeh(even(+nat)) \\ m2 : modeh(odd(+nat)) \\ m3 : modeb(even(+nat)) \\ m4 : modeb(odd(+nat)) \\ m5 : modeb(+nat = s(-nat)) \end{array} \right. \qquad E = \left\{ \begin{array}{l} odd(s(s(s(s(s(0)))))) \\ not\ odd(s(s(0))) \\ not\ odd(s(s(s(s(0))))) \end{array} \right.$$

We assume the set I' of integrity constraints to restrict the language bias, which establishes that rules whose head is $even(X)$ or $odd(X)$ cannot have in the body $even(X)$ or $odd(X)$ literals. The final I is the union of I' and I_f. The set M of mode declarations is transformed into the top theory $\top$ given in Figure 1. The instantiated abductive task $\langle E, B \cup \top, I, \{rule/2, prule/2\}\rangle$ accepts then as a possible solution the set Δ translated into the inductive hypothesis H as follows:[1]

$$\Delta = \left\{ \begin{array}{l} rule(1, [(m2, [], []), (m5, [], [1]), (m3, [], [2])]) \\ rule(2, [(m1, [], []), (m5, [], [1]), (m4, [], [2])]) \end{array} \right. \qquad H = \left\{ \begin{array}{l} odd(X) \leftarrow X = s(Y), even(Y) \\ even(X) \leftarrow X = s(Y), odd(Y) \end{array} \right.$$

In the abductive search, the standard Prolog selection rule is adopted that selects clauses in order of appearance in the program. Since no head of clause in B unifies with the positive examples, the derivation uses one of the rules defined in $\top$. The selection of the *body* literal from the rule results in four derivation branches in the search tree, one for each of the four "body" clauses whose head unifies with it. A partial abductive hypothesis is generated, equivalent to the rule $odd(X) \leftarrow X = s(Y), even(Y)$. At this point, the condition $even(s(s(s(s(0)))))$, part of the current goal, is not entailed by B so one of the rules in $\top$ is used. It can be seen as an "artificial" example conditional to the partial hypothesis. The derivation results in the creation of an additional rule in the final hypothesis that defines the predicate *even*. The computation continues, thus excluding inconsistent hypotheses and those that entail also negative examples resulting in the final Δ. Partial rules are derived and used throughout the search so they can be referenced to define concepts that depend on them. It is also interesting to observe that the search is guided by the examples and thus only significant solutions are explored. The inductive solution H for this inductive problem is either not found by other ILP systems like PROGOL, or derived after a "blind"

[1] *prule* abducibles are omitted for brevity.

search as discussed in Sec. 5. The learning is non-observational (i.e. the *even* predicate is not observed in the examples). TAL is also able to learn the base case of the recursion. If the fact $even(0)$ is deleted from B and the mode declaration $modeh(even(\#nat))$ is added to M, TAL returns three solutions with the same set of examples: the first has the same definition of odd as in H and defines $even(s(s(s(s(s(0))))))$ as base case, the second and the third are the same as in H with $even(s(s(0)))$ and $even(0)$ respectively as base cases.

4. A case study

We employ a case study to compare TAL with the only other system capable of solving the same class of ILP problems XHAIL[2]. The following case study, taken from [Ray09a], represents a simple model of metabolic regulation for the bacterium E. coli and includes a formulation of the Event Calculus [Sha99] formalism. The target predicate *happens* is used to characterise the bacterium feeding mechanism based on the availability of sugar. See [Ray09a] for a more extensive explanation of the example.

$$B = \left\{ \begin{array}{l} \text{[Type definitions and Event Calculus axioms]} \\ initiates(add(G), available(G), T) \leftarrow sugar(G), timex(T) \\ terminates(use(G), available(G), T) \leftarrow sugar(G), timex(T) \\ happens(add(lactose), 0) \\ happens(add(glucose), 0) \end{array} \right. \quad I = \left\{ \begin{array}{l} \leftarrow happens(use(G), T), \\ not\ holdsAt(available(G), T) \end{array} \right.$$

$$E = \left\{ \begin{array}{l} holdsAt(available(lactose), 1) \\ holdsAt(available(lactose), 2) \\ not\ holdsAt(available(lactose), 3) \end{array} \right. \qquad M = \left\{ \begin{array}{l} m1 : modeh(happens(use(\#sugar), +timex)) \\ m2 : modeb(holdsAt(\#fluent, +timex)) \\ m3 : modeb(not\ holdsAt(\#fluent, +timex)) \end{array} \right.$$

The transformations in Definition 2.2 are applied to the given ILP instance. The abductive solution for the corresponding ALP problem is:

$$\Delta = \left\{ \begin{array}{l} rule(1, [(m1, [glucose], []), (m2, [available(glucose)], [1])]) \\ rule(2, [(m1, [lactose], []), (m2, [available(lactose)], [1]), (m3, [available(glucose)], [1])]) \end{array} \right.$$

equivalent to the inductive hypothesis:

$$H = \left\{ \begin{array}{l} happens(use(glucose), T) \leftarrow holdsAt(available(glucose), T) \\ happens(use(lactose), T) \leftarrow holdsAt(available(lactose), T), not\ holdsAt(available(glucose), T) \end{array} \right.$$

As discussed in [Ray09a], XHAIL generates *Kernel Sets* that serve as lower bound for the final hypothesis, through iterations of increasing in size abductive explanations until a satisfactory solution is found. Intuitively, a Kernel Set is computed in two phases. A first abductive phase finds the set Δ of the head of the rules in the Kernel Set and a second deductive phase constructs the body of the rules by computing all the ground instantiations of the body mode declarations that are implied by $B \cup \Delta$. Kernel Sets are generalised in a final inductive phase. Instead, TAL explores candidate solutions in a top-down manner, backtracking whenever the current solution leads to failure in the abductive derivation. The

[2]Relying on the results reported in [Ray09a], the computation time for this study appears to differ by one order of magnitude. [Ray09a] reports that a prototype XHAIL implementation took a couple of seconds to compute H on a 1.66 GHz Centrino Duo Laptop PC with 1 GB of RAM, while TAL took 30 ms to find H and 180 ms to explore the whole space of hypotheses limited to two clauses with at most two conditions on a 2.8 GHz Intel Core 2 Duo iMac with 2 GB of RAM. Unfortunately, XHAIL is not publicly available so we are not able to perform an empirical comparison of the performance of the two systems.

partial hypotheses are already in their final form and are implicitly tested for correctness whenever a new example is selected in the abductive derivation.

5. Discussion and related work

We have implemented TAL in YAP Prolog [Cos08] using a customised implementation of the ASYSTEM [Kak01] that integrates the SLDNFA proof procedure with constraint solving techniques. TAL has been tested on various non-monotonic ILP problems like the examples proposed in [Kim09], [Alr09] and [Ray07]. It has also been used to perform Theory Revision [Wro96], i.e. to change, according to some notion of minimality, an existing theory [Cor09]. This work is is motivated by the project [Ban08] that seeks to exploit the proposed approach in the context of learning privacy policies from usage logs. We performed preliminary experiments in this direction applying an extended version of TAL to the Reality Mining dataset [Eag06] where examples of refused calls were used to learn general rules. A score based on accuracy and complexity of the rules was employed to prune the search space.

The idea of a top theory as bias for the learning has been initially introduced in TOPLOG [Mug08], which performs deductive reasoning on the background knowledge extended with the top theory. Candidate hypotheses are derived from single positive examples and then the best ones are selected after a hill climbing search. SPECTRE [Bos94] also requires a user-provided overly general theory that is specialised by unfolding clauses until no negative examples are covered. HYPER [Bra99], specialises an overly general theory, deriving rules that are subsumed by those in the theory. Thus the number of rules in the final hypotheses cannot increase. FOIL and related approaches like [Coh94] perform an informed hill climbing search. These systems are not fully non-monotonic since they disallow negation in the background knowledge or in the hypotheses. In contrast to TOPLOG, TAL generates candidate hypotheses also considering negative examples, excluding *a priori* solutions that entail some of the negative examples. In general, we regard TAL a generalisation of other top-down ILP systems. Constraining it to consider only positive examples, the background theory and mode declarations being definite, would result in the same rule generation mechanism as TOPLOG. Different search strategies can be easily implemented by modifying the abductive procedure. Partial solutions can be associated with a score with respect to the examples (e.g. the sum of entailed examples over the total). This would enable the use of informed search techniques and strategies like, for instance, hill climbing or beam search that can be used to prune the space, exploring the most promising solutions. Similarly to TOPLOG [Mug08], our approach can also be applied directly to a grammar based language bias specification, instead of generating the top theory from mode declarations. Systems based on Inverse Entailment (IE), compute a bottom clause, or set of clauses (e.g. the Kernel Set) that constrains the search space from the bottom of the generality lattice. For problems dealing with definite theories our system manages to solve a wider class of problems than PROGOL, since one single example can generate more than one rule in H. IMPARO [Kim09] solves a class of problems whose solutions are not found by other IE based systems, namely connected theories where body conditions are abductively proved from the background theory. These problems, that are solved in Imparo by applying *Induction on Failure* (IoF), can also be solved by TAL, as shown in the example given in this paper. The IoF mechanism is in our system embedded in the abductive search that always includes in the search space the generation of a new rule whenever a condition is not entailed by the current theory. XHAIL can find hypotheses computed under IoF by exploring non-minimal

abductive explanations but the search is not guided by the background theory and a partial hypothesis[3]. This highlights another advantage of TAL: the computation of clause heads in the hypothesis is naturally interleaved with the generation of the body and it does not take place in a separate phase as in IMPARO and XHAIL. Moreover, all rules are constructed concurrently and their partial definitions can be used. [Kak00] propose a system for inductive learning of logic programs that compared to TAL is limited to observational predicate learning. Finally, [Adé95] introduces induction in the abductive SLDNFA procedure, defining an extended proof procedure called SLDNFAI. In contrast, TAL defines a general methodology and does not commit to a particular proof procedure. Moreover SLDNFAI does not allow a fine specification of the language bias, makes no use of constraints on the generated hypotheses and is limited to function-free definite clauses.

6. Conclusions and further work

We have presented a novel approach to non-monotonic ILP that relies on the transformation of an ILP task into an equivalent ALP task. We showed through an example how the approach is able to perform non-observational and multi-predicate learning of normal logic programs by means of a top-down search guided by the examples and abductive integrity constraints where a partial hypothesis is used in the derivation of new rules. In contrast, techniques based on IE perform a blind search or are not able to derive a solution. The mapping into ALP offers several advantages. Soundness and completeness can be established on the basis of the abductive proof procedure employed. Constraint solving techniques and optimised ALP implementations can be used and abductive integrity constraints on the structure of the rule can be employed. Furthermore, the search space makes use of partial hypotheses that allows the use of informed search techniques, thus providing a general framework that can scale to learning problems with large datasets and theories. We obtained promising result in this direction and we are currently evaluating the use of heuristics and informed search techniques. We plan to investigate the properties of the mapping and the relationships with the search space of other ILP techniques.

Acknowledgements

The authors are grateful to Krysia Broda, Robert Craven, Tim Kimber, Jiefei Ma, Oliver Ray and Daniel Sykes for their useful discussions. This work is funded by the UK EPSRC (EP/F023294/1) and supported by IBM Research as part of their Open Collaborative Research (OCR) initiative.

References

[Adé95] Hilde Adé and Marc Denecker. Ailp: Abductive inductive logic programming. In *IJCAI*, pp. 1201–1209. 1995.

[Alr09] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastián Uchitel. Learning operational requirements from goal models. In *ICSE*, pp. 265–275. 2009.

[Ban08] A. K. Bandara, B. A. Nuseibeh, and B. A. Price et al. Privacy rights management for mobile applications. In *4th Int. Symp. on Usable Privacy and Security*. Pittsburgh, 2008.

[3]In the "odd/even" example the only way for XHAIL to find the correct solution is to extend the minimal abductive solution $odd(s(s(s(s(s(0))))))$ with $even(s(s(s(s(0)))))$ and $even(s(s(s(s(s(s(0)))))))$ that have to be somehow chosen from a set of infinite candidates.

[Bos94] H. Boström and P. Idestam-Almquist. Specialization of logic programs by pruning SLD-trees. *GMD-Studien*, vol. 237. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[Bra99] Ivan Bratko. Refining complete hypotheses in ILP. In *ILP '99: 9th Workshop on Inductive Logic Programming*, pp. 44–55. Springer-Verlag, London, UK, 1999.

[Cla77] Keith L. Clark. Negation as failure. In *Logic and Data Bases*, pp. 293–322. 1977.

[Coh94] William W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artif. Intell.*, 68(2):303–366, 1994.

[Cor09] D. Corapi, O. Ray, A. Russo, A.K. Bandara, and E.C. Lupu. Learning rules from user behaviour. In *5th Artif. Intell. Applications and Innovations (AIAI 2009)*. Thessaloniki, Greece, 2009.

[Cos08] Vítor Santos Costa, Luís Damas, Rogério Reis, and Rúben Azevedo. Yap user's manual, 2008.

[DŎ0] Sašo Džroski. Relational data mining applications: an overview. pp. 339–360, 2000.

[Eag06] Nathan Eagle and Alex Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[Esp00] Floriana Esposito, Giovanni Semeraro, Nicola Fanizzi, and Stefano Ferilli. Multistrategy theory revision: Induction and Abduction in INTHELEX. *Mach. Learn.*, 38(1-2):133–156, 2000.

[Kak92] Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *J. Log. Comput.*, 2(6):719–770, 1992.

[Kak00] Antonis C. Kakas and Fabrizio Riguzzi. Abductive concept learning. *New Generation Comput.*, 18(3):243–, 2000.

[Kak01] C. Kakas, Antonis, Bert Van Nuffelen, and Marc Denecker. A-system : Problem solving through abduction. In *17th International Joint Conference on Artif. Intell.*, vol. 1, pp. 591–596. IJCAI, inc and AAAI, Morgan Kaufmann Publishers, Inc, 2001.

[Kim09] Tim Kimber, Krysia Broda, and Alessandra Russo. Induction on failure: Learning connected horn theories. In *LPNMR*, pp. 169–181. 2009.

[Lav94] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. 1994.

[Llo87] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[Mal98] D. Malerba, F. Esposito, and F. A. Lisi. Learning recursive theories with ATRE. In H. Prade (ed.), *Proc. of the 13th European Conference on Artif. Intell.*, pp. 435–439. John Wiley & Sons, 1998.

[Moy03] S Moyle. *An investigation into theory completion techniques in inductive logic*. Ph.D. thesis, University of Oxford, 2003.

[Mug95] S. Muggleton. Inverse entailment and Progol. *New Generation Computing J.*, 13:245–286, 1995.

[Mug08] Stephen Muggleton, José Carlos Almeida Santos, and Alireza Tamaddoni-Nezhad. Toplog: Ilp using a logic program declarative bias. In Maria Garcia de la Banda and Enrico Pontelli (eds.), *ICLP*, *LCNS*, vol. 5366, pp. 687–692. Springer, 2008.

[Qui96] J. Ross Quinlan. Learning first-order definitions of functions. *J. Artif. Intell. Res. (JAIR)*, 5:139–161, 1996.

[Ray07] Oliver Ray. Inferring process models from temporal data with abduction and induction. In *1st Workshop on the Induction of Process Models*. 2007.

[Ray08] Oliver Ray and Chris Bryant. Inferring the function of genes from synthetic lethal mutations. In *2nd Int. Conf. on Complex, Intelligent and Software Intensive Systems*, pp. 667–671. IEEE Computer Society, 2008.

[Ray09a] Oliver Ray. Nonmonotonic abductive inductive learning. In *Journal of Applied Logic*, vol. 7, pp. 329–340. 2009.

[Ray09b] Oliver Ray, Ken Whelan, and Ross King. A nonmonotonic logical approach for modelling and revising metabolic networks. In *3rd Int. Conf. on Complex, Intelligent and Software Intensive Systems*. IEEE Computer Society, 2009.

[Ric95] Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[Sha94] Leon Shapiro and Ehud Y. Sterling. *The Art of PROLOG: Advanced Programming Techniques*. The MIT Press, 1994.

[Sha99] Murray Shanahan. The event calculus explained. *LNCS*, 1600, 1999.

[Wro96] Stefan Wrobel. First order theory refinement. In Luc De Raedt (ed.), *Advances in Inductive Logic Programming*, pp. 14 – 33. IOS Press, 1996.

[Yam97] Akihiro Yamamoto. Which hypotheses can be found with inverse entailment? In *ILP*, pp. 296–308. 1997.